Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 57 (2015) 915 – 922
Procedia Computer Science

3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015)

Detection of File Level Clone for High Level Cloning

Manu Singh[a], Dr. Vidushi Sharma[b]
School of ICT,Gautam Buddha University, Yamuna Expressway, Greater Noida, India

Abstract

This paper presents description about structural clones and presents a hybrid approach for the detection of File level clone. Structural clones are similar program structures that are formed by lower level smaller clones with similar code fragments. Recurrent occurrence of simple clones in a file may lead to higher file level clones. The proposed approach detects high level clones in terms of file clones in different or same directories. The proposed approach uses the combination of text based and metric based analysis of a source code. This is done with the help of High Level Clone (HLC) detection tool designed in DOT NET. This approach is implemented as a common tool for input of various programming language and the existence of clone can be detected across the source code of different languages.

*Keywords:*Structural Clones; File level clone; method clone; higher level clone;simple clone; metrics.

1. Introduction

To keep pace with the ongoing change in requirement of technology and functions, there is a need to timely upgrade and maintains the system. As software is customized due to replication in code it becomes more and more complicated and difficult to maintain. This replication is sometimes called cloning in software and occurs at different levels of abstraction and may have different origin [1]. Several techniques have been proposed to detect similar clone fragments also called simple clones [2] but analysis of similarities at elevated levels of abstraction still remains a promising area. Literature shows that 50% of the source code is cloned [3]. Clone management remains far from industrial acceptance, and this area has gained more awareness now [4].High level clones may be used to take out important information about a software system design and implementation phase which may be additionally used to know about evolution, reuse and reengineering opportunities of software system [5]. In this paper, our aim is discuss the structural clones with emphasis on file level clone. The paper also shows the results of metric similarity analysis and simulation program to justify our detection system.

2. Related Work

In present time's High level clones (HLC) is a promising area that uses a hierarchical organization of lower level clone fragments (Simple clones) to form bigger level clones (High Level Clone). Many research groups classified clone with reference to different contexts. We have reviewed all such available classifications of clones in our earlier research work [6] and presented them in the form of a High Level Clone Classification. In this classification we classified it into method level clone, file level clone and directory level clone as structural

Peer-review under responsibility of organizing committee of the 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015)
doi:10.1016/j.procs.2015.07.509

clone. Structural clones are formed by lower level, smaller clones, with similar code fragments at the bottom of such hierarchy. This concept of moving from lower level similarities to higher level similarities can be repeatedly applied, leading to the discovery of design concepts at various levels of abstraction. A large amount work has been done on the detection of simple clones, but less has been done for detecting these higher level similarities.

In the literature several clone detection approaches have been projected for simple clones [7]. A basic classification of simple clone is identified: a) Approaches based on Text or String: In this source code is considered as sequence of string or lines. Two code fragments are compared with each other to search longest common subsequences of same string or text. b) Approaches based on token sequence: In this the whole source code is changed into a tokens sequence and scanned to locate cloned subsequences. c) Approaches based on AST: In this the whole source code is parse into an abstract syntax tree (AST) and detect similar subtrees. d) Approaches based on PDG: In this the whole source code is transformed into a Program Dependency Graph (PDG), on which an isomorphic subgraph matching algorithm is applied for searching similar subgraphs. e) Approaches based on metrics: metrics are used for code fragments; instead of comparing code directly metric vectors are compared. One of the approaches in literature detects High-Level Concept Clones in Source Code [8]. This approach analyzes the source code to identify comments and identifiers. Then detect similar high-level concepts (e.g., abstract data types). The approach uses latent semantic indexing to statically examine the software system and determine semantic similarities between source code documents (i.e., functions, files, or code segments). These measures of similarities are used in the clone detection approach but, in some cases, the developers choose to completely rename the data structure and operation names in a cloning of a list (e.g., a list of new records), then comments are also deleted, this technique is unable to detect such similarities. A novel clone detection technique [9] was proposed, which transform input source code into tokens and then compare source code token-by-token. They used suffix-tree- matching algorithm to compare, in which the clone detection is worked on the basis of searching the nodes on the tree but, this method is unable to detect clones with structures identification, identifiers regularization, clone measurement etc. A tool CCFinder is developed for its implementation, which detect code clones in COBOL, C, C++, Java source codes. Another approach is based on Language Independent Approach [10] for Duplicated Code Detection; Detecting techniques for duplicate code exist but process depends mostly on parsers. In this paper shows that it is promising to avoid this hindrance by applying a language independent and visual approach, i.e. a tool without requiring parsing, it is able to detect a major amount of code duplication. Mayrand et al. [11] use various metrics to detect clone. Functions with similar metrics values are identify as code clones. Metrics are calculated from names, layout, expressions, and control flow of functions. A function clone is identified as a pair of whole function bodies with similar metrics values.

There are numerous benefits that can be achieved from categorization of cloning but, there are various problems associated with cloning. Code cloning can increase unnecessary code in size. Cloning code can lead to idle, or unused, code in the system that left unchecked can cause problems with code clarity, understandability, readability, and maintainability of the software system. Maintenance cost and efforts can be increased when these problems or bugs have to be fixed multiple times, and these changes could be lead to errors. Code clones that are not well understood can introduce new bugs in the software system. For example, variable names may be shared and customized unintentionally. Understandability of program can be affected by the increased code size, as well as the requirement to understand the differences between the duplicates. Research and study on code cloning regularly asserts that duplicating code within a software system causes damage to the system's design and should be ignored.

There are numerous differences between our proposal and the approaches discussed above. The main distinctive feature of our methodology is the use of hybrid approach to determine file level cloned code. This system uses both the metric and textual analysis of a source code. This is done with the help of clone detection tool designed in .net called HLC detection tool and also discuss results with metrics calculation after verification on several small c programs.

Metric similarity analysis is used to identify potential clone candidates and line by line comparison of files is used to discard false positives.

3. METRICS BASED SIMILARITY ANALYSIS

Metrics based similarity analysis uses three source code metrics that are sensitive to several control and data flow program features. Metric values are computed for each file after filtrations and normalization.

The features examined for metric computation include:

1. FANOUT: The number of functions called
2. LOC: The total number of lines of code
3. McCabe Cyclometic complexity.

Metric value similarity analysis is based on the assumption that two code fragments S1 and S2 have metric value M(S1) and M(S2) for same source code metric M. If the two code fragments are similar under the set of features measured by M, then the values of M(S1) and M(S2) should be proximate. A description of the metrics used is given below but a more detailed description can be found in (Adamov, 1987) [12], (Fenton 1991) [13], (Moller,1993) [14] .

$$S\_COMPLEXITY(s) = FAN\_OUT(s)$$

Where FAN_OUT(s) is the number of individual function calls made within source code‘s’.

$$S\_LOC(s) = (LLN - FLN) + 1$$

Where

- LLN= Last line number
- FLN = First Line number

$$CC(s) = (1 + Ifs + LOOPs + CASEs)$$

Where

- CC(s) is the CYCLOMATIC COMPLEXITY(s) of source code
- Ifs are the number of if operators
- LOOPs are the number of loops in the source code
- CASEs are the number of switch brances (without default)

Alternatively Mc Cabe can be calculated using

$$McCABE(s) = \epsilon - n + 2$$

Where

- ϵ is the number of edges in the control flow graph
- n is the number of nodes in the graph

4. Structural Level Clone Detection System

The main purpose of the software system is to detect the high level similarity between two different files. For this purpose develop simulation program to detect file level clone. The novelty of approach is use of metrics based similarity analysis to finds out potential file clones. Files with the same metric value are identified as a potential clone. After deciding potential file clone, by comparison of selected potential file clone actual file clone can be detected line by line.

Structural clones may be viewed as collaborations of lower level (simple) clones. These clones may be defined as similar program that are formed by lower level, smaller clones, with similar code at the bottom of such hierarchy i.e. method level clone to file level clone and file level clone to directory level clone. The flowchart in figure 1 depicts the detection of structural clone.

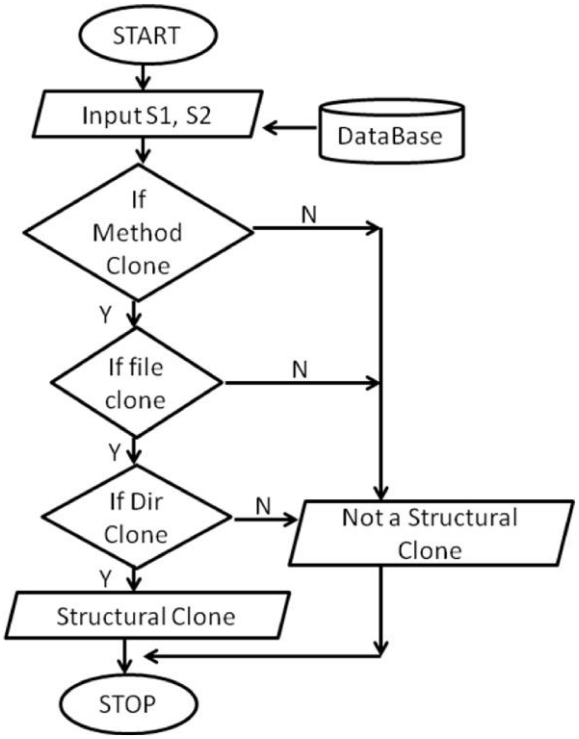

Figure 1: Detection of Structural Clone

4.1. DIRECTORY LEVEL CLONE

In the proposed system each file of a directory is compared with all the files in other directory to check whether they both are similar or not. After comparing all the files between both directories, if more than 80 percentage (Threshold value) of the files is similar then it is concluded that directory level cloning exists. Thresholds are mostly based on expert opinion.

4.2. METHOD LEVEL CLONE

In the proposed system source code of one file is given as input. Some metrics computed for this file are:

- M_FANOUT: The number of functions called
- M_LOC No. of Line of code in each method
- M_CC Complexity of each method

$$S\_COMPLEXITY(s)=FAN\_OUT(s)$$

Where FAN_OUT(s) is the number of individual function calls made within s.

$$M\_LOC(s)= (LLN – FLN) + 1$$

Where

- LLN= Last Line number
- FLN = First Line number

$$M\_CC(s)=(1 + Ifs +LOOPs +CASEs)$$

Where

- M_CC(s) is the CYCLOMATIC COMPLEXITY(s) of method.
- Ifs are the number of if operators
- LOOPs are the number of loops in the source code
- CASEs are the number of switch branches (without default)

The computed metric value of each method is compared if match is found between metric values of methods in two files then the method clone may exist between them. After comparing the metrics value each line of a method is compared with all the lines in other method to check whether they both are similar. if more than 80 percentage of the lines are similar then it is concluded that they both methods are clones but detection of method clone does not prove that file clone exist because there may be a situation that the method clone exist but file clone does not exist. After verifying the method clones there may be the possibility of file level cloning. The degree of method cloning may determine the probability of file level cloning.

4.3. FILE LEVEL CLONE

In the proposed system source code of two files are given as input to preprocessing. In preprocessing the statement which does not effect on analysis are removed like comment entries, preprocessors, white spaces, tabs etc. and files are converted into a structured format. Metrics computed for each file are:

- F_LOC(s) No. of Line of code in each file
- FAN_OUT Total No. of individual function in each file
- FCC(s) Complexity of each file

$$F\_LOC(s)= (LLN – FLN) + 1$$

Where

- LLN= Last line number
- FLN = First Line number

$$F\_COMPLEXITY(s)=FAN\_OUT(s)$$

Where FAN_OUT(s) is the number of individual function calls made within s.

$$FCC(s)=(1 + Ifs +LOOPs +CASEs)$$

Where

- FCC(s) is the CYCLOMATIC COMPLEXITY(s) of source code of each file

- Ifs are the number of if operators
- LOOPs are the number of loops in the source code

By using this approach detect the structural clone from lower level to higher level i.e. from method level clone to directory level clone. First we measure metrics for each method to detect potential method clone. If potential method clone exist then gradually increase the level for detection of file level clone. To detect potential file level clone compare some metrics for each file and then compare line by line similarity in potential file clone. Then gradually increase the level and each file of a directory is compared with all the files in other directory to check whether they both are similar or not. After completing comparing all the files between both directories, if more than 80 percentages of the files is similar then it is concluded that there is directory level cloning exist. At the end if similarity is found at all structural levels then it concludes that structural Clone exists.

5. File Level Clone Detection

We projected an approach for file level clone detection in proposed HLC higher level clone detection tool. In this approach input two files from database and normalize both files by deleting comments and white spaces and filter the files by removing header files, main function, getch function etc. Then compute LOC and complexity of each file. If values of both metics are same then compare both files line by line. Eighty percent similarity of both files conclude that file level clone exist in that directory.

6. Algorithm of File Level Clone Detection

Input: Two source codes S1, S2

Step 1 Inputting: Input S1, S2 // source codes of two files in which clones are to be detected.
Step 2 Normalizing: by deleting comments and white spaces
Step 3 Filtering: remove header files, main(), getch(), etc.
Step 4 Compute Metrics: compute LOC and Complexity of each file.
Step 5 Compare Metrics // detect potential clone
Step 6 Compare line by line similarity of source code if similar then goto step 9
Step 7 If line similarity does not exist then goto step 8
Step 8 Display Clone does not exist and goto step 10
Step 9 Check threshold value // If similarities are found to a particular threshold value, it means code contains File clones
Step 10 End

7. Results And Discussion

The proposed approach has been tested on self created tool with the input of C project files, JAVA project files, text files and C# project files. The metrics similarity analysis shows that similar methods are potential method clones but detection of method clone does not prove that file clone exist because there may be a situation that the method clone exists but file clone does not exist. After verifying the method clones there may be the possibility of file level cloning. The degree of method cloning may determine the probability of file level cloning.

The results are obtained based on the similarity between methods and files. A sample result is shown below in Table 1 and Table 2. This shows how much data in both methods and files is cloned and demonstrates clone existence between two files.

Table 1: Method level Metrcis Computation for both Files

Method ID	Method Fanout (M_FANOUT)	Method Line of Source Code (M_LOC)	Complexity (M_CC)	Potential method clone candidate
M1	2	20	5	CLONE
M2	3	18	4	
M1	2	20	5	NOT
M3	2	30	4	
M2	3	18	4	NOT
M3	2	30	4	

In the proposed system methods of file is given as input. Metric values are computed for each method. The computed metric value of each method is compared if match is found between metric values of methods in given one or two files then the method clone may exist between them. In this result M2 is equal to M1; because there is similarity exist between them at some threshold .i.e. eighty percent of threshold value.

Table 2 : File level Metrcis Computation for both Files

File ID	File Fanout (F_FANOUT)	Line of Source Code (F_LOC)	File Complexity (F_CC)	Potential File Clone Candidate
F1	3	100	4	NOT
F2	6	235	6	
F1	3	100	4	CLONE
F3	3	90	4	
F2	6	235	6	NOT
f3	3	90	4	

The computed metric value of each file is compared if match is found between metric values of files then the file clone may exist between them. In this result F3 is equal to F1; because there is similarity exist between them at some threshold .i.e. eighty percent of threshold value.

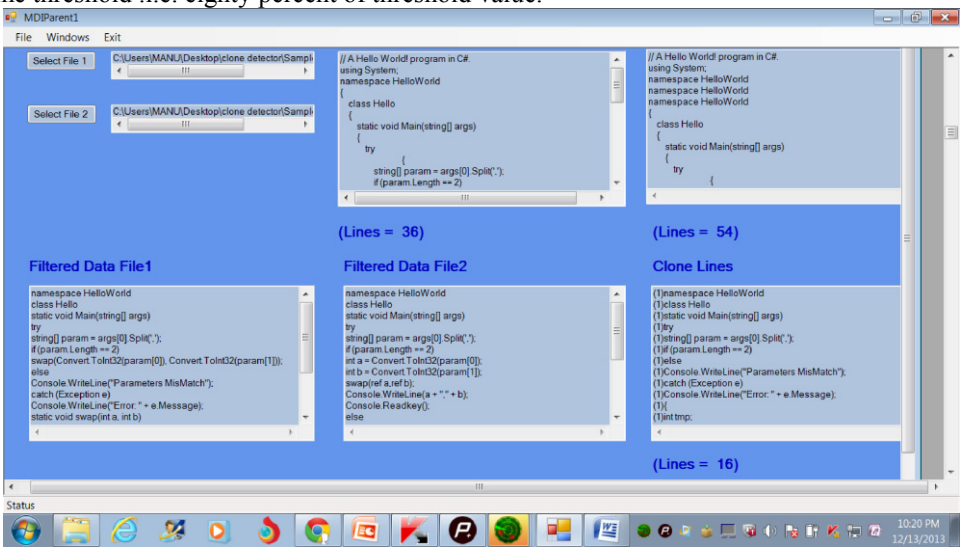

Figure 2: Detection of Cloning In Two Files

The similarity between files is measured through line by line comparison of the filtered form of the files and having the following two parameters, Number of cloned lines and the Total number of lines.

8. Conclusion

Code clone are similar program fragments that comes in various forms in software systems. Maintenance phase play an important role in software life cycle process. Software maintenance cost contributes major to the total development cost. We introduced the idea of higher-level similarities as a repeating configuration of lower-level clones. The algorithm starts by finding simple clones (that is, similar code fragments). Gradually move to higher-level similarities by using data mining technique. Our approach is both scalable and useful. Simple clones as well as high level clone information leads to better program understanding, helps in different maintenance related tasks, and points to potential reusable components. Higher level clones are also candidates for association with generic design solutions. After such association, programs are easier to understand, modify and reuse.

The threshold levels are adjustable as per the requirement, though standardization is required for determining the threshold value in future. This implemented system combines both the text based and metric based techniques for file level clone detection. Metric based technique is a straight forward one, so it is a light weight technique. The text based technique is the one which gives high precision. In addition to file level clone this work can also be extended by detecting the proposed directory level clones and method level clones in our future paper.

References

[1] H. A. Basit, S. Jarzabek, " A Case for Structural Clones", International Workshop on Software Clones (IWSC), 2009.
[2] William S. Evans , Christopher W. Fraser and Fei Ma, "Clone detection via structural abstraction" , Software quality journal Volume 17, Number 4, 2009.
[3] B. S Baker , "On finding duplication and near duplication in large software system" , proceedings of Second IEEE Working Conference on Reverse Engineering, 1995.
[4] M. Zibran, C. Roy, "The Road to Software Clone Management: A Survey", Tech. Report 2012-03, Department of Computer Science, University of Saskatchewan, Canada, pp. 1-62, 2012.
[5] Cory Kasper, Michael W. Godfrey, "Cloning considered harmful", Working Conference on Reverse Engineering, 2006
[6] M. Singh, V. Sharma, "High Level Clones Classification", International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-2, Issue-6, August 2013.
[7] C. Roy, J. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," Science of Computer Programming, vol. 74, pp. 470–495, 2009.
[8] A. Marcus., J. Maletic, "Identification of high-level concept clones in source code", In: ASE '01. 2001
[9] T. Kamiya, S. Kusumoto and K. Inoue, "CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code", IEEE Trans. Software Engineering, vol. 28, pp.654-670,2007
[10] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code", Proc. IEEE Int'l Conf. Software Maintenance, pp. 109-118, 1999.
[11] J. Mayrand, C. Leblanc and E. Merlo, " Ecperiment on the automatic detection of Function clones in Software System Using Metrics", in proceeding of the 12th International Conference on Software Maintenance (ICSM'96), pp. 244-253, Monterey, CA, USA, November 1996.
[12] R. Adamov, "Literature review on software metrics", Zurich: Institut fur Informatik der Universitat Zurich, 1987.
[13] E. Fenton, "Software metrics: a rigorous approach", Chapman and Hall, 1991.
[14] K. Moller, "Software metrics: a practitioner's guide to improved product development", 1993.